|  |
| --- |
| [公司名称] |
| VCFLIB说明文档 |
| [文档副标题] |

|  |
| --- |
| Oliver  2023-5-16 |

目录

[vcflib功能 1](#_Toc135164208)

[vcflib 目录结构 1](#_Toc135164209)

[编译方法 1](#_Toc135164210)

[1.依赖库 1](#_Toc135164211)

[2.操作步骤 1](#_Toc135164212)

[示例程序 test 2](#_Toc135164213)

[1.读操作流程 2](#_Toc135164214)

[2.读写操作流程 3](#_Toc135164215)

[示例程序 simpleRead 3](#_Toc135164216)

[vcflib 详细接口 4](#_Toc135164217)

[1.文件打开关闭接口 4](#_Toc135164218)

[2.文件读接口 5](#_Toc135164219)

[3.文件写接口 8](#_Toc135164220)

[4.其他接口 9](#_Toc135164221)

[vcflib 用户可调节的宏参数 11](#_Toc135164222)

## vcflib功能

该库主要功能是可以高速读取vcf文件进内存中。支持未压缩的vcf文件和zlib压缩过的gz格式的vcf文件。

## vcflib 目录结构

**Makefile：**该库的编译规则文件，通过make完成对该库的编译

**bin：**存放正确编译完毕后，生成的该库相关文件**（未编译前为空）**

vcflib.h:该库的头文件，其中有该库提供的所有函数接口的相关声明,及相关用户可调节的宏参数

libvcf.so:该库的动态库文件

libvcf.a: 该库的静态库文件

**inc：**存放该库源码的头文件

baseTool.h: 该库中用到的内部基本功能函数的声明

vcflib.h:该库的头文件，其中有该库提供的所有函数接口的相关声明,及相关用户可调节的宏参数

**obj：**存放该库在编译过程中产生的中间文件**（未编译前为空）**

**src：**存放该库的源码文件

baseTools.c：该库中用到的内部基本功能函数

vcfFile.c：该库中的主要功能接口函数

**tst：**存放使用该库的示例代码，包含一个test（读操作和一个读写操作）的示例程序和一个simpleRead（最简读操作）的示例程序。

## 编译方法

### 1.依赖库

请保证使用该库的环境中，已经正确安装如下软件库：

gcc zlib pthread openmp

### 2.操作步骤

进入vcflib目录

**cd vcflib**

执行编译命令，完成库的编译

**make**

如果编译成功，在vcflib/bin目录中会生成：**vcflib.h**（该库的头文件）、

**libvcf.so**（该库的动态库文件）、**libvcf.a**（该库的静态库文件）。可将生成的对应库文件拷贝到适当位置完成对该库的使用。

如果修改了该库中的源码需要重新编译

**make clean**

**make**

## 示例程序 test

在vcflib/tst目录中，提供了一个调用该库的测试程序。在成功编译该库后，进入该目录

**cd tst**

编译示例程序

**make test**

用户在使用该库时，编译参数可参考vcflib/tst/Makefile文件。

vcflib/tst/test.c示例程序演示了一个读操作和一个读写操作。下面对test.c文件做详细讲解（下面的line number都是test.c的行号）。

### 1.读操作流程

1）定义VCF\_FILE结构体文件指针变量，定义vcf行结构体变量，定义vcf多行（块）结构体变量，及其他相关变量。见line11-line15。

其中行结构体和块结构体变量第一次使用时必须清空，然后就可以重复使用了。

2）打开文件操作，见line23

vcfFileOpen(&fp\_read,old\_file\_path,FILE\_MODE\_GZ,P\_DS|P\_GT)

fp\_read为前面定义的文件指针；

old\_file\_path为要读的vcf文件路径，这里利用全局变量old\_file\_path提供；

FILE\_MODE\_GZ:表示输入的测试文件的格式为gz的压缩格式（支持非压缩格式FILE\_MODE\_NORMAL和压缩格式FILE\_MODE\_GZ）；

P\_DS|P\_GT:指本次读操作将同时解析DS和GT字段（若只解析GT字段本参数填写 P\_GT，当前只支持只解析GT和同时解析GT、DS两种模式）

该函数的详细功能见后面的**vcflib详细接口**章节的介绍。

3）读vcf文件头操作，见line30

vcfFileReadHead(&fp\_read)

其头部信息会存入fp\_read.head结构体对象中，该函数的详细功能见后面的**vcflib详细接口**章节的介绍。

对文件头操作支持如下功能：

获得vcf文件meta-information的行数；

获得vcf文件samples个数；

在指定位置添加meta-information；

删除指定位置meta-information；

见line36-line84.这些函数的详细功能见后面的**vcflib详细接口**章节的介绍。

4）读vcf文件中的markers。该库支持两种操作：

每次读1个marker（没有进行多线程加速，无速度优势）。见line87

vcfFileReadDataLine(&fp\_read,&dataLine)

将当前vcf文件的一个marker读入dataline结构体中，该函数的详细功能见后面的**vcflib详细接口**章节的介绍。

每次读n个markers（利用openMP进行多线程加速，有速度优势）。

vcfFileReadDataBlock(&fp\_read,&dataBlock,10)

将当前vcf文件的10个marker读入dataBlock结构体中，该函数的详细功能见后面的**vcflib详细接口**章节的介绍。

5）清空并释放vcf行结构体变量，vcf多行（块）结构体变量，见line109-line111。

6）关闭要操作的文件，见line113。

vcfFileClose(&fp\_read)

该函数的详细功能见后面的**vcflib详细接口**章节的介绍。

### 2.读写操作流程

1）配置VCF\_FILE结构体读对象和写对象，见line118。

2）用配置好的VCF\_FILE结构体读对象打开要读的文件，见line143。用配置好的VCF\_FILE结构体写对象创建要写的文件，见line149

3）读vcf文件头，将其读入fp\_read.head中，并在第一个位置和最后位置各添加一行自定义的meta\_information，写入新创建的vcf文件中。见line162-line172

4）循环读取源vcf文件中的1000个markers，并将其写入新的vcf文件中。直至源文件全部被读完。见line176-line204.

5）释放临时内存空间、块内存空间，关闭两个文件，见line207-line212。

## 示例程序 simpleRead

在vcflib/tst目录中，提供了一个调用该库的测试程序。在成功编译该库后，进入该目录

**cd tst**

编译示例程序

**make simpleRead**

用户在使用该库时，编译参数可参考vcflib/tst/Makefile文件。

vcflib/tst/simpleRead.c示例程序演示了一个最简读操作。下面对simpleRead.c文件做详细讲解（下面的line number都是simpleRead.c的行号）。

1）引用了必要的vcflib的头文件，见line1。

2）定义并赋值了输入文件路径的变量，见line2。

3）定义VCF\_FILE结构体文件指针变量，定义vcf行结构体变量，定义vcf多行（块）结构体变量，及其他相关变量。见line6-line8。其中行结构体和块结构体变量第一次使用时必须清空，然后就可以重复使用了。

4）打开vcf文件操作，见line9。

5）读vcf文件头操作，见line15。

6）读一行vcf的marker到dataLine结构体中，并利用printDataLine函数展示一下该行内容，见line17-line18。

7）读10行（一块）vcf的markers到dataBlock结构体中，该函数的第三个参数指定该块有多少个markers。推荐使用这种方式读数据，有openmp的并行加速优势。利用printDataLine函数展示一下该块的首行和尾行内容。见line 21 – line 25。

8）清空并释放vcf行结构体变量，vcf多行（块）结构体变量，见line27-line28。

9）关闭操作的vcf文件，见line30。

## vcflib 详细接口

### 1.文件打开关闭接口

**VCF\_STATUS vcfFileOpen(VCF\_FILE \*fp,const char \*fileName, FILE\_MODE fileMode, unsigned int parseItem);**

**功能：**

以读的方式打开一个存在的vcf文件，支持未压缩的文本文件格式，支持zlib压缩的gz格式。并开启vcflib日志

**参数：**

fp：结构体VCF\_FILE的指针。vcflib库中的文件handle，结构体定义如下：

typedef struct

{

union

{

FILE \* fp; //未压缩格式文件指针

gzFile gfp; //zlib压缩格式文件指针

}fp;

FILE\_MODE mode; //存储第3个参数fileMode的内容

//记录该文件的压缩格式

//FILE\_MODE\_NORMAL :未压缩

//FILE\_MODE\_GZ: zlib压缩gz格式

FILE\_HEAD head; //用于存储该vcf文件的文件头数据

//在未作读文件头操作之前，内容为空

int numSamples; //该vcf文件的sample个数

//在未作读文件头操作之前，该值为0

//读过文件头后，该值为文件的sample个数

unsigned int parsingItems; //用户设置解析marker中的字段选项bit

//P\_GT是解析GT字段(当前为默认解析)

//P\_DS是解析DS字段

}VCF\_FILE;

fileName：要打开的vcf文件路径及名称

fileMode：指定打开文件的压缩格式，FILE\_MODE\_NORMAL表示未压缩；FILE\_MODE\_GZ表示zlib压缩gz格式

parseItem：指定对文件进行读操作时，解析哪些字段。P\_DS|P\_GT表示读操作时将同时解析DS和GT字段；P\_GT表示读操作时将只解析GT字段

**返回值：**

VCF\_ERROR：打开失败

VCF\_OK：打开成功

**VCF\_STATUS vcfFileCreate(VCF\_FILE \*fp,const char \*fileName,FILE\_MODE fileMode);**

**功能：**

以写的方式创建一个vcf文件。若文件已经存在则清空其中内容。支持未压缩的文本文件格式，支持zlib压缩的gz格式。并开启vcflib日志

**参数：**

fp：结构体VCF\_FILE的指针；详解见上（**vcfFileOpen**）

fileName：要打开的vcf文件路径及名称

fileMode：指定打开文件的压缩格式，FILE\_MODE\_NORMAL表示未压缩；FILE\_MODE\_GZ表示zlib压缩gz格式

**返回值：**

VCF\_ERROR：创建/打开失败

VCF\_OK：创建/打开成功

**VCF\_STATUS vcfFileAppend(VCF\_FILE \*fp,const char \*fileName,FILE\_MODE fileMode);**

**功能：**

以写的方式创建一个vcf文件。若文件已经存在则将保留其中内容，并将文件写指针定位到文件尾。支持未压缩的文本文件格式，支持zlib压缩的gz格式。并开启vcflib日志

**参数：**

fp：结构体VCF\_FILE的指针；详解见上（**vcfFileOpen**）

fileName：要打开的vcf文件路径及名称

**返回值：**

VCF\_ERROR：追加/打开失败

VCF\_OK：追加/打开成功

**VCF\_STATUS vcfFileClose(VCF\_FILE \*fp);**

**功能：**

关闭已经打开的文件，清空并释放文件头所占用的内存空间。并关闭vcflib日志

**参数：**

fp：结构体VCF\_FILE的指针；详解见上（**vcfFileOpen**）

**返回值：**

VCF\_ERROR：关闭失败

VCF\_OK：关闭成功

### 2.文件读接口

**VCF\_STATUS vcfFileReadLine(VCF\_FILE \*fp,char \*lineStr,int lineSize);**

**功能：**

从打开的fp所指向的vcf文件中，读取当前行，将内容存入lineStr中。

**参数：**

fp：结构体VCF\_FILE的指针

lineStr：存储读取文件内容的内存指针

lineSize：lineStr所指向的内存长度

**返回值：**

VCF\_ERROR：读取失败

VCF\_OK：读取成功

**VCF\_STATUS vcfFileReadHead(VCF\_FILE \*fp);**

**功能：**

从打开的fp所指向的vcf文件中，读取vcf文件头部分，将内容存入fp->head所指向的结构体中。

**参数：**

fp：结构体VCF\_FILE的指针,其中的成员变量head为结构体FILE\_HEAD类型的变量。用于存储该vcf文件的文件头，结构体定义如下：

typedef struct

{

char \*\*metaInfoLines; //用于存储vcf文件头的meta information line

int numMetaInfoLines; //vcf文件头的meta information line行数

char \*headerLine; //用于存储vcf文件头的header line

}FILE\_HEAD;

**返回值：**

VCF\_ERROR：读取失败

VCF\_OK：读取成功

**VCF\_STATUS getNumMetaInfoLines(FILE\_HEAD \*fhp,int \*NumMetaInfoLines);**

**功能：**

从fhp所指向的vcf文件头中，获取meta information line行数

**参数：**

fhp：结构体FILE\_HEAD的指针。详解见上（**vcfFileReadHead**）

NumMetaInfoLines：所得到的fhp所指向的vcf文件头中meta information line行数

**返回值：**

VCF\_ERROR：读取失败

VCF\_OK：读取成功

**VCF\_STATUS getNumSamples(VCF\_FILE \*fp,int \*NumSamples);**

**功能：**

从打开的fp所指向的vcf文件中，获取该文件的sample个数

**参数：**

fp：结构体VCF\_FILE的指针

NumSamples：所获得的sample个数

**返回值：**

VCF\_ERROR：读取失败

VCF\_OK：读取成功

**char\* vcfFileParseDataLineInfo(char \*lineStr,DATA\_INFO \*dataInfo);**

**功能：**

将lineStr内的一行vcf文件的数据中的前九项，解析到dataInfo指向的DATA\_INFO结构体中。操作后lineStr中去掉前九项

**参数：**

lineStr：vcf文件的一行数据

dataInfo：指向的DATA\_INFO结构体指针。vcflib库中用于存储vcf文件中每行数据的前九项，结构体定义如下：

typedef struct

{

char \*chrom;

char \*pos;

char \*ID;

char \*ref;

char \*alt;

char \*qual;

char \*filter;

char \*info;

char \*format;

}DATA\_INFO;

**返回值：**

去掉前九项的vcf文件数据行

**VCF\_STATUS vcfFileParseDataLine(VCF\_FILE \*fp,char \*lineStr,DATA\_LINE \*dlp);**

**功能：**

将lineStr内的一行vcf文件的数据，解析到dlp指向的DATA\_LINE结构体中

**参数：**

fp：结构体VCF\_FILE的指针

lineStr：vcf文件的一行数据

dlp：指向的DATA\_LINE结构体指针。vcflib库中用于存储vcf文件中每行数据，结构体定义如下：

typedef struct

{

char \*rawDataLine; //该行数据的原串首地址，不是整字符串

//其中前九项的\t已经被替换成\0

DATA\_INFO dataInfo; //该行数据的前九项

char \*samplesRawString; //该行数据去掉前九项后的数据内容原串

char \*gtData; //该行数据GT部分数据

float \*dsData; //该行数据DS部分数据

int numSamples; //该行数据Sample的个数

}DATA\_LINE;

**返回值：**

VCF\_ERROR：解析失败

VCF\_OK：解析成功

**VCF\_STATUS vcfFileReadDataLine(VCF\_FILE \*fp,DATA\_LINE \*dlp);**

**功能：**

从打开的fp所指向的vcf文件中，读取当前数据行，将内容存入dlp指向的DATA\_LINE结构体中

**参数：**

fp：结构体VCF\_FILE的指针

dlp：指向的DATA\_LINE结构体指针。详解见上（**vcfFileParseDataLine**）

**返回值：**

VCF\_ERROR：读取失败

VCF\_OK：读取成功

**VCF\_STATUS vcfFileReadDataBlock(VCF\_FILE \*fp,DATA\_BLOCK \*dbp,int numLines);**

**功能：**

从打开的fp所指向的vcf文件中，读取numLines个数据行，将内容存入dbp指向的DATA\_BLOCK结构体中

**参数：**

fp：结构体VCF\_FILE的指针

dbp：指向的DATA\_BLOCK结构体指针。vcflib库中用于存储vcf文件中每行数据，结构体定义如下：

typedef struct

{

DATA\_LINE \*dataLines; //存储该数据块的多行数据的数组指针

int numDataLines; //该数据块数据行数

}DATA\_BLOCK;

numLines：从文件中一次性读取的行数

**返回值：**

VCF\_ERROR：读取失败

VCF\_OK：读取成功

**VCF\_STATUS vcfFileReadDataBlockOverlap1Line(VCF\_FILE \*fp,DATA\_BLOCK \*dbp,int numLines);**

**功能：**

从打开的fp所指向的vcf文件中，读取numLines个数据行，将内容存入dbp指向的DATA\_BLOCK结构体中。除第一次以外，后面每次读取的数据块的第一行内容都是上次读取的数据块中最后一行内容。即有一行overlap

**参数：**

fp：结构体VCF\_FILE的指针

dbp：指向的DATA\_BLOCK结构体指针。详解见上（**vcfFileReadDataBlock**）

numLines：从文件中一次性读取的行数

**返回值：**

VCF\_ERROR：读取失败

VCF\_OK：读取成功

### 3.文件写接口

**VCF\_STATUS vcfFileWriteLine(VCF\_FILE \*fp,char \*lineStr);**

**功能：**

将lineStr内容，做为一行写入打开的fp所指向的vcf文件中

**参数：**

fp：结构体VCF\_FILE的指针

lineStr：存储写入文件内容的内存指针

**返回值：**

VCF\_ERROR：写入失败

VCF\_OK：写入成功

**VCF\_STATUS vcfFileWriteHead(VCF\_FILE \*fp,FILE\_HEAD \*fhp);**

**功能：**

将fhp所指向的结构体中的vcf文件头信息，写入打开的fp所指向的vcf文件中

**参数：**

fp：结构体VCF\_FILE的指针

fhp：结构体FILE\_HEAD的指针

**返回值：**

VCF\_ERROR：写入失败

VCF\_OK：写入成功

**VCF\_STATUS vcfFileAddMetaInfoLine(FILE\_HEAD \*fhp,int posIndex,char \*MetaInfoLine);**

**功能：**

将MetaInfoLine内的meta information line增加到fhp所指向的vcf文件头结构体中，并指定新增加的meta information line位置在meta information中的第posIndex行

**参数：**

fhp：结构体FILE\_HEAD的指针

posIndex：指定写入meta information中的第几行

MetaInfoLine：要写入的meta information内容

**返回值：**

VCF\_ERROR：写入失败

VCF\_OK：写入成功

**VCF\_STATUS vcfFileRemoveMetaInfoLine(FILE\_HEAD \*fhp,int posIndex);**

**功能：**

将fhp所指向的vcf文件头结构体中，第posIndex条meta information line删除

**参数：**

fhp：结构体FILE\_HEAD的指针

posIndex：要删除meta information中的第几行

**返回值：**

VCF\_ERROR：删除失败

VCF\_OK：删除成功

### 4.其他接口

**void clearFileHead(FILE\_HEAD \*fhp);**

**功能：**

清空fhp所指向的FILE\_HEAD结构体中的数据并释放其内存空间

**参数：**

fhp：结构体FILE\_HEAD的指针

**返回值：**

无

**void clearDataLine(DATA\_LINE \*dlp);**

**功能：**

清空dlp所指向的DATA\_LINE结构体中的数据并释放其内存空间（注：若调用vcfFileReadDataLine接口后，用相同dlp参数再次调用vcfFileReadDataLine接口之前，需要调用本接口清空并释放上一次DATA\_LINE结构体中的内容，否则会造成内存泄露）

**参数：**

dlp：结构体DATA\_LINE的指针

**返回值：**

无

**void clearDataBlock(DATA\_BLOCK \*dbp);**

**功能：**

清空dbp所指向的DATA\_BLOCK结构体中的数据并释放其内存空间

**参数：**

dbp：结构体DATA\_BLOCK的指针

**返回值：**

无

**VCF\_STATUS vcfPopSubString(char \*\*lineStr,char \*subStr);**

**功能：**

从lineStr所指向的字符串中，截出用空格、制表符‘\t’、换行符‘\n’所分隔的第一段子串，存入subStr中。操作后lineStr中去掉截出的子串

**参数：**

lineStr：待截断的长字符串

subStr: 截出来的子串

**返回值：**

VCF\_ERROR：截取失败；或待截长字符串为空（长字符串全部截完）

VCF\_OK：截取成功

**void printDataLine(DATA\_LINE \*dlp);**

**功能：**

打印显示dlp指向的DATA\_LINE结构体中的所有成员变量的值，用于显示dlp中存储内容。

**参数：**

dlp：结构体DATA\_LINE的指针

**返回值：**

无。

## vcflib 用户可调节的宏参数

在inc/vcflib.h文件中，有一些宏定义，用户可以根据不同的需求进行动态调整。

**BT\_MAX\_LINE\_SIZE：**当前该库支持的读取vcf文件中，一行中最大的字符数量。当前值为6M。若因需要处理的vcf文件一行字符数特别大而产生了错误，可以将其值改大后重新编译即可。

**OPEN\_MP\_THREAD\_NUM：**在进行读取vcf文件中的数据解析时，开启的openMP的线程数。当前值为10。用户可以提高该值来提高整个库读取vcf文件的速度，但整体速度受硬盘I/O限制和CPU核数限制。

**（注：调节后需要重新编译该库才会生效）**